

# Ready-Mode Receive: An Optimized Receive Function for MPI

Ron Brightwell

Scalable Computing Systems, Sandia National Laboratories, Albuquerque, NM 87185-1110,  
USA,  
bright@cs.sandia.gov

**Abstract.** This paper describes an extension to the MPI Standard that offers the potential to increase performance for receiving messages. This extension, the *ready-mode receive*, is the receive-side equivalent of the ready-mode send. This paper describes the semantics of this new receive function and discusses the potential for performance improvement that it offers. In particular, we discuss how the current trend toward using intelligent network interfaces may increase the potential for significant performance improvement.

## 1 Introduction

In February of 1996, the author made an informal proposal to other members of the MPI Forum for a new receive function that offered the opportunity for increased performance. At the time, the MPI Forum was working on additions and changes to version 1.0 of the MPI Standard [1], which became MPI version 2.0 [2]. The initial reaction to this proposal was largely negative, so a more formal proposal with supporting evidence was never pursued. However, we believe that the proposed receive function warrants further investigation for several reasons. In this paper, we describe the proposed function and discuss the possible benefits it offers. We also discuss some of the arguments that were presented against including the function in the Standard.

The rest of this paper is organized as follows. The following section describes the new receive function in detail and discusses the possible performance implications surrounding it. Included in this section is a summary of the initial arguments against this function from other members of the Forum. Section 3 continues with a rebuttal of these arguments and presents some further motivations for investigating the possible benefits of the new receive operation. We conclude by summarizing the key points of this paper in Section 5 and provide an outline of future activities in Section 4.

## 2 The Ready-Mode Receive

The MPI version 1.2 Standard defines four different modes for sending data. Each of these modes has different semantics and associated with moving data that affect how the underlying MPI implementation handles buffering and completion. The *standard-mode* send operation has no semantic guarantees to buffering or completion. The MPI implementation is free to use any buffering strategy (e.g. send-side or receive-side) and

completion may or may not be tied to the existence of a matching posted receive at the destination process. A *buffered*-mode send is also free to complete independent of activity at the receiver, but in this case, the sending process has given the MPI implementation sufficient space to buffer the outgoing message should it need to. The *synchronous*-mode send does not complete until a matching receive operation has been initiated at the destination. In addition to data movement, this send mode provides an explicit synchronization point between the processes. Finally, the *ready*-mode send may not be started until a matching receive has already been posted at the destination. This mode offers the opportunity for increased performance. Since the receive is guaranteed to be posted, the sender can make optimizations with respect to avoid buffering and avoiding handshake protocols with the receiver.

The ready-mode send is the only mode whose semantics are driven by performance. It is an indication from the application programmer of an opportunity to avoid protocol and buffering overhead in the interest of increased performance. The possible benefits of ready-mode is fairly straightforward. Eliminating intermediate buffering, especially for large messages, saves resources and avoids the performance degradation of memory-to-memory copies. Avoiding handshaking protocols eliminates the extra overhead of communicating with the destination process before moving the data. To some extent, these costs are relatively easy to measure.

In contrast to modes for sending, there are no equivalent receive modes defined by MPI for the semantics associated with message reception. The semantics of a receive operation are determined by the matching send operation. For example, a receive operation that matches with a synchronous mode send needs to generate an acknowledgment to the sender in order for the send to complete. A receive operation that matches with a standard mode send may not need to do anything to complete the send. Since the semantics of standard-mode, buffered-mode, and synchronous-mode are all defined by message completion, it is appropriate for the receive operation to determine what if anything needs to be done after a matching message has been found.

However, this is not the case for the receive operation that matches a ready-mode send. For this operation, the semantic is defined by message initiation. That is, the ready-send cannot start until a matching receive has been posted. Similar to the optimization opportunity of a ready-mode send, the corresponding receive operation that matches a ready-mode send has a possible opportunity for optimization. Since the semantics of ready-mode send guarantee that a matching receive has been posted at the destination, the matching receive operation is guaranteed that no matching message has yet arrived. For a *ready-mode receive*, there is no need to search a queue of unexpected messages for a possible match.

## 2.1 Potential Benefits

The obvious potential benefit of the ready-mode receive is avoiding the time needed to search an unexpected message queue before posting the receive. This time can be dependent on several factors, but primarily depends on the average length of the unexpected queue and the cost associated with traversing it. This information is highly dependent on the MPI implementation and the message passing structure of the application. The performance increase may be significant for MPI implementations where

the cost of searching an queue is relatively high or for applications whose unexpected queue can grow relatively long. Unfortunately, no real data for these two measurements is readily available or published.

For some implementations, searching an unexpected queue may have side effects beyond the time wasted to perform the operation. For example, some MPI implementations (e.g. [3, 4]) maintain two structures for maintaining an unexpected message queue. Part of the queue is maintained by the network interface and part is maintained inside the MPI implementation. For our MPI implementation for Portals 3.0 [4], events related to unexpected messages must be maintained in two separate queues. Message receive events are placed in a queue by the network interface and consumed by the MPI implementation. If MPI consumes an event from this queue that does not match the current receive operation, it must hold on to this event in a separate queue. This could potentially result in several unneeded memory-to-memory copies of message header information and possibly data.

Since searching an unexpected queue and posting a receive must be an atomic operation, the MPI library must be extremely careful to retain atomicity. As such, the time needed to post a receive is dependent on the frequency of message arrival. If a message arrives after the unexpected queue is search, but before the receive is posted, this new message must be checked to insure that it does not match the receive. If messages are continually coming in while a standard receive is being posted, the post will be delayed until all of the receives have been made visible to MPI and the network has quiesced.

The performance of a ready-mode receive should be deterministic. Adding an entry to a posted receive queue should take a fixed amount of time. This is unlike the standard receive mode, which must traverse an arbitrarily long queue and possibly exchange protocol messages with the sender. This deterministic behavior may be beneficial to applications where the exchange of messages is more tightly synchronized, such as in some soft real-time applications. Deterministic performance of posting a receive may also enhance performance debugging by making detection of performance anomalies easier.

Since the ready-mode receive is a fundamental operation, implementations which do not have support for it can simply use the standard receive operation. The ready-mode receive offers an optimization opportunity, but does not sacrifice portability or correctness for those implementations that do not take advantage of the opportunity.

## 2.2 Drawbacks

A proposal for a ready-mode receive function was never formally brought to the MPI Forum. Initial feedback on the MPI mailing list was largely negative, with a few members adamantly opposed to such a function. Several reasons were cited.

First, the opportunity for performance is not readily evident or easily quantifiable. No data was available to support the claim that the ready-mode receive could provide a significant performance improvement. It was believed that high-performance applications, at least those that would be concerned about saving microseconds by not searching a queue, would have relatively short unexpected message queues. And, if the unexpected queue does grow to a point where search time would be significant, the MPI implementation should consider using a hashing function to avoid a linear search. This

approach would increase the time to insert the unexpected into the queue, but would reduce the time needed to search the queue.

It was also pointed out that ready-mode sends are an optimization intended to address long messages, so saving a few microseconds for a communication that takes orders of magnitude longer is of little or no gain.

As for deterministic performance, it could be argued that for applications whose communication patterns are relatively consistent, the number of unexpected messages is likely to be low. An application that would benefit from the deterministic performance of a ready-mode receive is unlikely to have an unexpected queue long enough to cause a large variance in the time needed to search the queue.

Aside from the technical reasons related to performance, there were other reasons not to consider the ready-mode receive function. Those members of the MPI-2 Forum who participated in the MPI-1 Forum indicated that the ready-mode send mode was accepted into the Standard by a very close vote. Many on the Forum did not see any real performance benefit for ready-mode, and in an effort to try to keep the number of functions for data movement minimal, did not support it in the first place. The ready-mode receive function was viewed as a further optimization for a mode that was largely unsupported.

Lastly, it was argued that an additional receive function would be too confusing for application developers. In 1996, application developers had limited experience with MPI, and any semantic change to the basic data movement operations was viewed as an additional obstacle for application developers. The obvious drawback is that incorrect use of the ready-mode receive function would cause non-compliant programs and result in undefined behavior.

### 3 Current Motivations

At this point we have discussed the potential benefits and drawbacks of a ready-mode receive mode. We believe some of the original arguments against have weakened over time.

The important question of the performance benefits still remains open. The need for empirical evidence remains. We believe that the need for investigation of the ready-mode receive has grown due to the evolving networking technology to which the current generation of MPI implementations is targeted.

Commodity cluster computing has displaced proprietary parallel systems as the most popular platform for high-end scientific and engineering computing. Gigabit networking technology that utilizes intelligent or programmable network interface cards, including Myrinet [5], Quadrics [6], and VIA [7], have characteristics that may increase the cost of unnecessarily traversing the MPI unexpected queue. These network cards currently all use a PC's PCI bus, which is a significant bottleneck to achieving network performance. High-performance message passing layers typically avoid crossing the PCI bus whenever possible. Unnecessarily searching the MPI unexpected queue may involve a significant amount of traffic on the PCI bus. This may be especially true for implementations that offload MPI functionality onto the network interface card, a

practice which is becoming more common as the computational power and memory capacity of network interfaces continues to increase.

We also believe that advanced MPI implementations will continue to move more data movement functionality from the user-space library down to intelligent or programmable networking hardware. As this happens, applications seeking the highest levels of performance will migrate toward those functions in MPI that offer greater opportunity for optimization, such as the ready-mode send and persistent communication mode.

In our experience developing and supporting MPI implementations for large-scale parallel machines, we have seen applications that require a significant number of unexpected messages. In fact, we have had to enhance an implementation solely on the basis of being more flexible in supporting larger numbers of unexpected messages, especially as applications are scaled up to thousands of processes. The number of unexpected messages increases with the number of processes in the job. We have seen applications that have exceeded 1024 unexpected messages on only a few hundred processors. Perhaps it can be argued that such applications are poorly designed or structured. It may be precisely this type of application that could benefit from restructuring using ready-mode send and receive functions.

In addition to direct usage by applications, the ready-mode receive function has many possible uses in supporting current MPI functionality or other possible extensions to the Standard.

One possible use is in the MPI-2 one-sided operations. A possible portable implementation of the `MPI_Win_get()` function would be to post a receive and send a request message to the target. In this case, there would be no reason to search the unexpected queue to see if a matching message has already arrived. This optimization may decrease the latency of the get operation, especially in the case of a non-blocking get.

Some collective operations may be able to avail of the ready-mode receive as well. For example, `MPI_Allreduce` and `MPI_Reducescatter` functions may be staged so such a receive needs to be posted before a request or contribution of data is given. Avoiding a search of the unexpected queue in this case may not provide any performance improvement since all collectives are currently blocking operations. Should future versions of MPI support non-blocking collectives, a ready-mode receive may be more effective. Nevertheless, since the ready-mode receive is a more fundamental operation, it may provide other benefits beyond raw performance to these collective operations.

## 4 Future Work

We believe that the ready-mode receive function may offer performance gains for applications using the ready-mode send. We also believe that unnecessarily searching the MPI unexpected queue may have resource management side effects that may ultimately affect performance and/or effective use of resources. We intend to explore both of these issues in more depth and hope to provide experimental results that can be used to determine the effectiveness of the proposed receive operation.

Ideally, we would be able to find a real-world application that can be used to characterize the performance benefits of the ready-mode receive. However, we also intend to gather data related to the average length of the MPI unexpected message queue for different applications on various numbers of nodes.

A standard API or extension for gathering low-level performance data from within an MPI implementation is currently being developed as part of a research project sponsored by the United States Department of Energy's Accelerated Strategic Computing Initiative (ASCI). This work is being carried out in collaboration with MPI Software Technology, Inc., Pallas GmbH, and Intel KAI. This portable interface is intended to provide access to low-level performance data inside an MPI implementation, such as the length of the unexpected queue and the time a message has waited in the unexpected queue. We hope to leverage this interface to gather particular information that relates to the ready-mode receive.

## 5 Summary

In this paper, we have proposed an extension of the MPI Standard to support a new receive function, the ready-mode receive. This new function provides an opportunity for improving performance by avoiding the unnecessary traversal of an MPI unexpected message queue. We have discussed many advantages of this new function and also presented many arguments that do not support it. We believe that more in-depth analysis is needed to adequately evaluate the effectiveness of a ready-mode receive function. We intend to use a new interface that is being developed for gathering low-level MPI implementation data to more fully understand the implications and impact of this new receive function.

## References

1. Message Passing Interface Forum: MPI: A Message-Passing Interface standard. The International Journal of Supercomputer Applications and High Performance Computing **8** (1994)
2. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. (1997) <http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html>.
3. Dimitrov, R., Skjellum, A.: An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing. In: Proceedings of the Third MPI Developers' and Users' Conference. (1999) 15–24
4. Brightwell, R., Maccabe, A.B., Riesen, R.: Design and Implementation of MPI for Portals 3.0. (Submitted for Consideration to EuroPVM/MPI 2002)
5. Boden, N., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.: Myrinet-a gigabit-per-second local-area network. IEEE Micro **15** (1995) 29–36
6. Petrini, F., chun Feng, W., Hoisie, A., Coll, S., Frachtenberg, E.: The Quadrics Network: High-Performance Clustering Technology. IEEE Micro **22** (2002) 46–57
7. Compaq, Microsoft, and Intel: Virtual Interface Architecture Specification Version 1.0. Technical report, Compaq, Microsoft, and Intel (1997)